REMARKS

The Office Action mailed December 11, 2008, considered and rejected claims 1-3 and 5-56. Claims 1-3 and 5-6 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Tim *Biernat* (*Persistent Object ID Service – Part 1,* IBM, June 2, 2003, pp. 1-6) in view of *Sun* (*Using the Timer Service, Sun Microsystems,* December 4, 2003, pp. 1-9), *Hibernate* (HIBERNATE - *Relational Persistence for Idiomatic Java, hibernate,* October 13, 2003, pp. 1-123), Tim *Biernat* (*Persistent Object ID Service – Part 2: Architecture and Design on the Server Side,* IBM, August, 2003, pp. 1-21) (hereinafter *Biernat-2*), and Ceki *Gülcü* (*Short Introduction to log4j,* March, 2002, pp 1169). Claims 7-11 were rejected under 35 U.S.C. § 103(a) as being unpatentable over *Hibernate* in view of Kaoru *Yoneyama* (U.S. Patent No. 6,985,892) and *Biernat-2*. Claims 12-17, 19-26 and 28-29 were rejected under 35 U.S.C. § 103(a) as being unpatentable over *Hibernate* in view of *Yoneyama, PostgreSQL (PostgreSQL 7.4.23 Documentation*). Claims 18 and 27 were rejected under 35 U.S.C. § 103(a) as being unpatentable over *Hibernate* in view of *Yoneyama*, *PostgreSQL*, and *Sun*. Claims 30-56 were rejected under 35 U.S.C. § 103(a) as being unpatentable over *Biernat* in view of *Hibernate* and *PostgreSQL*.[1]

By this response, claims 1, 7, 8, 10, 11-13, 15, 16, 21, and 32-52 are amended, claims 30, 31, and 53-56 are canceled, and claim 57 is added.[2] Claims 1-3, 5-29, 32-52, and 57 remain pending of which claims 1, 7, 12, and 57 are independent.

Overview of the Invention and Claim Amendments

The present invention is directed to embodiments for providing an object generator base class that includes the functionality to perform incrementation (i.e incrementing a property value of an object that is generated at each execution of the generator). By having the incrementation functionality in a base class, the programmer can create a generator (or tool) by only providing code that performs the object generation. The actual incrementation of the property value is abstracted from the programmer and handled by the base class. *See* Pg. 4, lines 15-16. For example, a programmer may wish to create a test tool that generates multiple files using the same

[1] Although the prior art status of the cited art is not being challenged at this time, Applicant reserves the right to challenge the prior art status of the cited art at any appropriate time, should it arise. Accordingly, any arguments and amendments made herein should not be construed as acquiescing to any prior art status of the cited art.

[2] Support for the amendments may be found primarily on page 3 and 7 of the specification. For example, the object generation method in the claims refers to the function labeled (3) on page 3, line 25.

operation with each file having a different name. The present invention allows the programmer to simply write the code to perform the single operation (in the claims, this is referred to as the object generation method) and let the base class handle incrementing the name of the file for each execution. This makes it much easier for the programmer to create generators to perform an iterative custom task. All the programmer has to do is provide the code that is repeatedly called to generate the objects, while allowing the base class to vary the value of the property at each execution of the code. Each of the independent claims has been amended similarly to clarify this aspect of the invention. Claim 1, for example, is drawn to a computer-readable medium that stores the base generator class. Claims 7 and 12 are drawn to methods of executing a generator that uses the base generator class to perform the incrementation. Claim 57 is a computer-readable medium claim for performing the same method as in claim 12.

Section 103 Rejections

Each of the claims was rejected as being obvious in view of the cited references as detailed above. Applicant submits, however, that these references fail to teach or suggest each limitation of the independent claims as currently amended. In general terms, these references fail to teach or suggest providing a framework consisting of a base class that abstracts incrementation from the inheriting class that provides the code to generate the objects.

Biernat is directed to embodiments for retrieving persistent object IDs (POIDs) in a distributed object system. Generating a POID is distinct from generating objects that include a property that is varied for each object. A POID is a long not an object. *See* Section entitled Design (stating that the actual value type of the POID is primitive long). As a long, the POID cannot have a property assigned to it. Additionally, although each POID is unique, Biernat does not address how they are generated. In contrast, Biernat only addresses how to develop the client side service which requests them. Therefore, because Biernat generates longs rather than objects, and does not address any aspect of how a POID is generated at the server to be sent to the client, it fails to teach or suggest any of the limitations.

Biernat-2 describes the same architecture as Biernat. Therefore, the same arguments regarding the POID being a long are equally applicable. On the other hand, Biernat-2 addresses the server side components that are used to create and manage POIDs. However, Biernat-2 discloses that the developer must create the code to perform the incrementation of the POID. *See* Section entitled EJB Design (stating that "the session bean exposes a method getBaseId() that contains the logic to locate or create the appropriate entity, increment its POID value by blocksize and return the next value."). Therefore, Biernat-2 fails to teach or suggest "a method that is overridden by an object generation method of the new generator class, the object generation method defining the single operation that generates an object at each iteration of the single operation;" and "a generator properties class that provides incrementation capability, which allows the value of the generator property of each generated object to vary during consecutive executions of the object generation method of the new generator class, such that the new generator class need not provide incrementation capability," as claimed in claim 1 in combination with the remaining limitations. The same arguments apply to the similar limitations which appear in independent claims 7, 12, and 57.

Hibernate discloses embodiments for mapping java classes to database tables. The cited section (4.1.4.1) discloses the use of a java class for generating unique identifiers which is very similar to what is disclosed in the Biernat references. Similar to these references, the unique identifiers of Hibernate are not objects but longs, shorts, or ints. Further, Hibernate only discloses that a single class generates the identifiers – there is no discussion of how this is done. Therefore, Hibernate also fails to teach or suggest the claimed limitations.

Yoneyama discloses the use of a file generator for generating a name for each image taken with a camera. However, Yoneyama only discloses that a single subroutine is used to generate the name. *See* Col. 12. Nothing is disclosed about how this subroutine is created or structured. A subroutine of this type is an example of the type of generator that can be created using the present invention. However, the present invention is not directed to simply creating a generator for incrementing a property of generated objects. For example, as is stated in the Background, "[s]oftware developers have written tools to satisfy the need to perform similar actions multiple times." Pg. 1, lines 13-14. However, "[r]equiring a tool developer to generate code for these functionalities each time the developer writes a tool program is not an efficient use of time and resources." Pg. 2, lines 11-13. The present invention, therefore, is directed to relieving the developer from having to generate code for performing incrementation. This is accomplished by providing a base class that provides the incrementation functionality while having "a method that is overridden by an object generation method of the new generator class [for generating the objects]." Because a developer only needs to provide the code to perform the object generation, the same base class may be used to provide incrementation for objects of any type (such as files or SQL requests). Yoneyama, and the remaining references, do not disclose these aspects.

Finally, Sun was only cited as teaching a schedule class, PostgreSQL was only cited as teaching the step and offset aspects of claims 12 and 57, while Gulcu was only cited as teaching the logging class. These references do not address generating objects with incremented values as claimed.

In summary, because none of the references address abstracting the incrementation functionality in a base class, these references do not render the claimed limitations obvious.

In view of the foregoing, Applicant respectfully submits that the other rejections to the claims are now moot and do not, therefore, need to be addressed individually at this time. It will be appreciated, however, that this should not be construed as Applicant acquiescing to any of the purported teachings or assertions made in the last action regarding the cited art or the pending application, including any official notice. Instead, Applicant reserves the right to challenge any of the purported teachings or assertions made in the last action at any appropriate time in the future, should the need arise. Furthermore, to the extent that the Examiner has relied on any Official Notice, explicitly or implicitly, Applicant specifically requests that the Examiner provide references supporting the teachings officially noticed, as well as the required motivation or suggestion to combine the relied upon notice with the other art of record.

In the event that the Examiner finds remaining impediment to a prompt allowance of this application that may be clarified through a telephone interview, the Examiner is requested to contact the undersigned attorney at (801) 533-9800.

Dated this 11th day of March, 2009.

Respectfully submitted,

Brian Tucker

RICK D. NYDEGGER
Registration No. 28,651
BRIAN D. TUCKER
Registration No. 61,550
Attorneys for Applicant
Customer No. 047973

RDN:BDT:gd
2287534_1.DOC